Programare Logică – Temă Colectivă de Laborator*

Claudia MUREŞAN
c.muresan@yahoo.com, cmuresan@fmi.unibuc.ro, claudia.muresan@unibuc.ro
Universitatea din Bucureşti, Facultatea de Matematică şi Informatică

2020–2021, Semestrul II

Exerciţiul 1. Pe o stradă sunt 5 case (aşezate pe o singură parte a străzii, adică pe un singur rând). Fiecare e zugrăvită într–o culoare diferită. În fiecare casă locuieşte un om venit dintr–o altă ţară, împreună cu animale de casă dintr–o singură specie. Fiecare locatar are o băutură preferată şi fumează o singură marcă de ţigări. Ştim că:

- englezul locuieşte în casa roşie;
- câinele aparţine spaniolului;
- locatarul casei verzi preferă să bea cafea;
- ucrainianul preferă să bea ceai;
- casa verde este la dreapta celei albe;
- fumătorul de Old Gold creşte melci;
- locatarul casei galbene fumează Kool;
- locatarul casei din mijloc preferă să bea lapte;
- norvegianul locuieşte în prima casă din stânga;
- fumătorul de Chesterfield este vecin cu proprietarul vulpii;
- fumătorul de Kool este vecin cu proprietarul calului;
- fumătorul de Gitanes preferă să bea vin;
- japonezul fumează Craven;
- norvegianul este vecin al casei albastre.

Să se determine:

① cine preferă să bea apă;

② cine creşte zebra şi în ce casă locuieşte (a câta din stânga şi de ce culoare).

*Obligatorie pentru ambele serii: de trimis într–un singur exemplar de fiecare grupă/semigrupă a seriei 31 şi de întreaga serie ID şi prezentat la laborator.

Indicaţie: folosiţi o listă a caselor, cu elemente de forma:

(culoare, naţionalitate, animal de casă, ţigări, băutură),

şi predicate auxiliare pentru testarea apartenenţei unui element la o listă şi pentru testarea apartenenţei a două elemente vecine la o listă, cu şi fără stabilirea ordinii celor două elemente în listă.

Notă: În următoarele exerciţii vom folosi convenţia din documentaţia pentru Prolog: argumentele precedate de + trebuie furnizate predicatelor în interogări, iar cele precedate de − vor fi construite de acele predicate.

Exerciţiul 2. Să se implementeze ciurul lui Eratostene pentru determinarea listei numerelor naturale prime mai mici decât un număr natural N, sub forma unui predicat binar $ciur(+Numar, -ListaNrPrime)$ care, dacă primeşte un număr natural $Numar$ ca prim argument, construieşte în al doilea argument lista numerelor naturale prime mai mici sau egale cu $Numar$, apelând următoarele predicate auxiliare:

un predicat binar $genlistanr(+Numar, -ListaNr)$ care, dacă primeşte un număr natural $Numar$ ca prim argument, generează în al doilea argument lista numerelor naturale mai mari sau egale cu 2 şi mai mici sau egale cu $Numar$;

un predicat binar $ciuruieste(+ListaNr, -ListaNrPrime)$ care apelează predicatul ternar de mai jos $filtreaza(+Nr, +Lista, -ListaFiltrata)$ cu primul argument dat de capul listei $ListaNr$ şi al doilea argument dat de coada acestei liste, astfel că $ListaNr = [Nr|Lista]$, reţine capul Nr al acestei liste drept cap al listei $ListaNrPrime$, apoi se apelează recursiv pentru primul argument dat de lista $ListaFiltrata$ produsă de predicatul $filtreaza$ şi, din acest apel recursiv, produce coada listei $ListaNrPrime$;

un predicat ternar $filtreaza(+Nr, +Lista, -ListaFiltrata)$ care, dacă primeşte un număr natural Nr ca prim argument şi o listă de numere întregi $Lista$ drept al doilea argument, produce în al treilea argument lista obţinută prin eliminarea multiplilor lui Nr din lista $Lista$.

Să se implementeze şi un predicat $ciur\_fis(+Numar, -ListaNrPrime, +ExtensieFisier)$, care să apeleze predicatul $ciur(+Numar, -ListaNrPrime)$, apoi să scrie $ListaNrPrime$ într–un fişier având numele 'primele_pana_la_$Numar$.$ExtensieFisier$', unde $Numar$ si $ExtensieFisier$ sunt valorile acestor argumente ale predicatului $ciur\_fis$ (folosiţi predicatele predefinite *write*, *tell* şi *told* pentru fişiere create şi deschise pentru citire, apoi închise, şi predicate pentru conversii între numere, atomi şi şiruri de caractere şi concatenare de atomi sau şiruri de caractere).

Exerciţiul 3. Să se scrie în Prolog un predicat $linlist(+Lista, -ListaLiniarizata)$ care "liniarizează" listele de liste, adică, dacă primeşte o listă $Lista$ ca prim argument, construieşte în al doilea argument lista obţinută prin înlocuirea fiecărui element de tip listă L al listei $Lista$ cu o sublistă formată din elementele listei L şi iterarea acestui procedeu, pentru elementele de tip listă ale lui L ş.a.m.d.; de exemplu, la interogarea:

?- $linlist([a, 1, [], f(X), [b, 2], g(X, f(1)), [c, [3, [], [d, f(5), 4, [5, e, 6]]], [7]]], Llin)$.

Prologul să răspundă: $Llin = [a, 1, f(X), b, 2, g(X, f(1)), c, 3, d, f(5), 4, 5, e, 6, 7]$.

Exerciţiul 4. Considerăm problema turnurilor din Hanoi: având trei tije şi un număr (natural nenul) de N inele discoidale cu diametrele de $1, 2, \ldots, N$ centimetri, respectiv, dacă aceste inele sunt aşezate pe prima tijă crescător după diametre de sus în jos, să se mute toate inelele pe a doua tijă, folosind–o pe a treia ca tijă de manevră şi respectând, la fiecare mutare a unui inel, regula ca inelul să nu fie mutat deasupra unui inel de diametru mai mic. Soluţia recursivă este: dacă notăm tijele cu a, b, c şi le reprezentăm ca stive, atunci, pentru orice $n \in \overline{1, N}$, pentru a muta inelele $i_1, i_2, \ldots, i_n$ de pe tija a pe tija b, mutăm, recursiv, inelele $i_1, i_2, \ldots, i_{n-1}$ de pe tija a pe tija c, apoi inelul i_n de pe tija a pe tija b, apoi, recursiv, inelele $i_1, i_2, \ldots, i_{n-1}$ de pe tija c pe tija b.

Să se scrie un predicat $hanoi(+N)$ care ilustrează grafic acest algoritm: dacă primeşte numărul N de inele ca argument, atunci, notând inelele cu numerele $1, 2, \ldots, N$ date de diametrele lor, să scrie pe

ecran aşezarea acestor inele pe cele 3 tije la fiecare pas al aplicării acestui algoritm, marcând cu "." tijele goale, cu o întârziere de o secundă după aplicarea fiecărui pas, folosind predicatul unar predefinit $sleep(+NumarSecunde)$.

De exemplu, pentru $N = 3$, afişările de la cei 8 paşi vor fi următoarele, dar aşezate unele sub altele:

```
1                                                                   1
2           2                       1           1                   2
3   .   .   3   1   .   3   1   2   3   .   2   .   3   2   1   3   2   1   3   .   .   3   .
```

Exerciţiul 5. Într–un fişier .PL să se includă baza de cunoştinţe GENERATII.PL ataşată acestei teme şi să se scrie un predicat binar $arbgen(+Persoana, -ArboreBinarGenealogic)$ care să fie satisfăcut ddacă $ArboreBinarGenealogic$ este arborele genealogic al persoanei $Persoana$, adică are eticheta rădăcinii $Persoana$, arborele genealogic al tatălui persoanei ca subarbore stâng şi arborele genealogic al mamei persoanei $Persoana$ ca subarbore drept, fiecare dintre aceşti subarbori fiind înlocuiţi cu arborele vid *nil* dacă părintele respectiv nu apare în baza de cunoştinţe, iar, pe lângă construirea acestui arbore genealogic, acest predicat să producă reprezentarea grafică a acestui arbore pe ecran, similară celei realizate de predicatul *desenarbbin* din fişierul ARBORI.PL din una dintre lecţiile de laborator, dar modificată astfel: eticheta rădăcinii arborelui să fie precedată de "−", etichetele fiilor drepţi să fie precedate de "/", iar etichetele fiilor stângi să fie precedate de "\".

Exerciţiul 6. Să se scrie două predicate ternare $detsubterm(+ListaNrNaturale, +Termen, -Subtermen)$ şi $detsubarb(+ListaNrNaturale, +Arbore, -Subarbore)$, care să fie satisfăcute ddacă $ListaNrNaturale$ este o listă de numere naturale nenule $N1, N2, \ldots, Nk$ şi:

- $Termen$ este un termen Prolog, iar $Subtermen$ este al Nk–lea subtermen al ... celui de–al $N2$–lea subtermen al celui de–al $N1$–lea subtermen al lui $Termen$;
- $Arbore$ este un arbore oarecare, reprezentat, ca în una dintre lecţiile de laborator, sub forma $arb(EtichetaRadacina, ListaSubarboriRadacina)$, iar $Subarbore$ este al Nk–lea subarbore al ... celui de–al $N2$–lea subarbore al celui de–al $N1$–lea subarbore al lui $Arbore$.

Exerciţiul 7. Să se scrie în Prolog:

- declaraţii pentru un operator unar prefixat '∼', de precedenţă 500, şi patru operatori binari infixaţi '−>','v','^','<−>', de precedenţă 700;
- considerând următoarele semnificaţii pentru aceşti operatori:

operator	conector logic
∼	negaţie logică
−>	implicaţie logică
v	disjuncţie logică
^	conjuncţie logică
<−>	echivalenţă logică

un predicat binar $conectprim(+Enunt, -EnuntScrisNumaicuNegatiesiImplicatie)$, care, într–o interogare în Prolog, dacă primeşte ca prim argument un enunţ în logica propoziţională clasică scris ca termen cu operatorii de mai sus drept conectori logici şi constante Prolog ale căror nume încep cu litere ca variabile propoziţionale, produce în al doilea argument acelaşi enunţ scris numai cu conectori logici primitivi; de exemplu, la interogarea:

?- conectprim(∼((p ^ ∼ r) −> ∼ ((p <−> r) v q)), EnuntModif).

Prologul să răspundă:

EnuntModif = ∼ (∼ (p−> ∼ (∼r))−> ∼ (∼ (∼ ((p−>r)−> ∼ (r−>p)))−>q)).

Observaţi că nu e nevoie să descompuneţi enunţurile compuse, care sunt date de termeni Prolog neconstanţi, cu predicatul "=.."; e suficient să folosiţi unificarea.

Exerciţiul 8. Să se scrie în Prolog două predicate binare $arbbinexpr(+Termen, -ArboreBinar)$ şi $termarbbin(-Termen, +ArboreBinar)$, care să fie satisfăcute ddacă $Termen$ este un termen Prolog în care nu apar operatori de aritate strict mai mare decât 2, iar $ArboreBinar$ este arborele binar asociat acestui termen ca arbore de expresie;
de exemplu, cu operatorii declaraţi la Exerciţiul 7, la interogările:

?- arbbinexpr(∼((p ^ ∼ r) −> ∼ ((p <−> r) v q)), Arbore).

?- termarbbin(Termen, arb(∼, arb(−>, arb(^, arb(p, nil, nil), arb(∼, arb(r, nil, nil), nil)), arb(∼, arb(v, arb(<−>, arb(p, nil, nil), arb(r, nil, nil)), arb(q, nil, nil)), nil)), nil)).

Prologul să răspundă, respectiv:

- Arbore = arb(∼, arb(−>, arb(^, arb(p, nil, nil), arb(∼, arb(r, nil, nil), nil)), arb(∼, arb(v, arb(<−>, arb(p, nil, nil), arb(r, nil, nil)), arb(q, nil, nil)), nil)), nil).
- Termen = ∼((p ^ ∼ r) −> ∼ ((p <−> r) v q)).

Exerciţiul 9. Cu operatorii declaraţi la Exerciţiul 7 şi enunţurile în logica propoziţională clasică scrise ca în Exerciţiul 7, să se scrie în Prolog un predicat binar

$$arbconectprim(+ArboreBinarEnunt, -ArboreBinarEnuntScrisNumaicuNegatiesiImplicatie),$$

care să transforme în mod direct, deci numai prin operaţii pe arbori binari, fără a construi termenii asociaţi acestora, arborele binar asociat unui enunţ în arborele binar asociat aceluiaşi enunţ scris numai cu conectori logici primitivi, astfel că, având predicatele implementate la Exerciţiile 7 şi 8, $arbconectprim(Arb, ArbModif)$ să fie satisfăcut ddacă există enunţurile, $Enunt$ şi $EnuntModif$ care satisfac conjuncţia:

$$termarbbin(Enunt, Arb),\ conectprim(Enunt, EnuntModif),\ arbbinexpr(EnuntModif, ArbModif).$$

Exerciţiul 10. Cu operatorii declaraţi la Exerciţiul 7 şi enunţurile în logica propoziţională clasică scrise ca în Exerciţiul 7 cu orice conectori logici, să se scrie în Prolog un predicat de aritate 4

$$eval(+ListaVarProp, +ListaValoriAdevarVarProp, +Enunt, -ValoareAdevarEnunt),$$

care să fie satisfăcut ddacă $ListaVarProp$ este o listă de constante ale căror nume încep cu litere, $Enunt$ este un enunţ scris ca termen Prolog cu aceste constante ca variabile propoziţionale, $ListaValoriAdevarVarProp$ este o listă de valori de adevăr pentru aceste variabile propoziţionale, iar $ValoareAdevarEnunt$ este valoarea de adevăr a enunţului $Enunt$ într–o interpretare care dă variabilelor propoziţionale din $ListaVarProp$, respectiv, valorile de adevăr din $ListaValoriAdevarVarProp$, şi care să calculeze valoarea de adevăr a enunţului în modul următor, folosind un predicat auxiliar binar declarat ca dinamic

$$h(+Enunt, -ValoareAdevarEnunt),$$

ale cărui clauze aflate în permanenţă în baza de cunoştinţe calculează valoarea de adevăr a unui enunţ recursiv pornind de la valorile de adevăr ale variabilelor propoziţionale care apar în acel enunţ:

dacă $ListaVarProp = [p1, p2, \ldots, pn]$, iar $ListaValoriAdevarVarProp = [Val1, Val2, \ldots, Valn]$, atunci $eval(+ListaVarProp, +ListaValoriAdevarVarProp, +Enunt, -ValoareAdevarEnunt)$ să adauge la baza de cunoştinţe, cu *assert*, faptele $h(p1, Val1)$, $h(p2, Val2)$, ..., $h(pn, Valn)$, apoi să apeleze predicatul $h(+Enunt, -ValoareAdevarEnunt)$, care să efectueze calculul valorii de adevăr a enunţului într–o interpretare care dă variabilelor propoziţionale $p1, p2, \ldots, pn$, respectiv, valorile de adevăr $Val1, Val2, \ldots, Valn$, iar, după efectuarea acestui calcul, predicatul *eval* să retragă din baza de cunoştinţe clauzele $h(p1, Val1)$, $h(p2, Val2)$, ..., $h(pn, Valn)$.